Modular Security Policy Design based on Extended Petri Nets

Hejiao Huang, Helene Kirchner

Modular Security Policy Design based on Extended Petri Nets

Hejiao Huang
Harbin Institute of Technology, China
INRIA Bordeaux Sud-Ouest, France
Email: hjhuang@hitsz.edu.cn
Hélène Kirchner
INRIA Bordeaux Sud-Ouest, France
Email: Helene.Kirchner@inria.fr

Abstract—Security policies are one of the most fundamental elements of computer security. Their design has to cope with composition of components in security systems and interactions between them. Consequently, a modular approach for specification and verification of security policies is necessary and the composition of modules must consistently ensure fundamental properties of security policies, in a rigorous and systematic way. This paper shows how to use extended Petri net process (EPNP) to specify and verify security policies in a modular way. It defines a few fundamental policy properties, namely completeness, termination, consistency and confluence, in Petri net terminology and relates them to classical notions. According to XACML combiners and to property preserving Petri net process algebra (PPPA), several policy composition operators are specified and property preserving results are stated for the policy correctness verification. The approach is illustrated on the design of a complex policy.

Index Terms—security policy, extended Petri net, specification and verification, property preservation.

1 Introduction

In our information age, the world's economy, communications, entertainment etc. depend on computers which are often connected by networks. Through networks, inestimably valuable information is transmitted, and hence security policies are required to protect the data (or other resources) from being processed by any undesirable users (subjects). Consequently, how to design highly dependable security policies that ensure secure access to distributed resources is an urgent problem.

As far as policy design is concerned, the following two requirements for a policy are the main source of difficulty and complexity:

- *Handling resources sharing and cooperation among heterogeneous systems*: a local policy is designed for handling a local request in a local system. A local system usually includes some private resources. In general, however, users' requests for services may be of very diversified nature, it is hardly feasible for a single local system to be able to contain enough resource information for supporting all kinds of services. A system should be capable of coordinating various resources and cooperating with other systems. Consequently, the global policy should be newly designed for handling the resources sharing and the cooperation among heterogeneous systems. This induces the difficult resource-sharing problem into the design of access control policies.
- *Component-based architecture*: the policies and resources of a system may be modeled, built or owned by different unrelated parties, at different times, and under different environments. Hence, in order to avoid severe interference in their individual developments, it is better for a policy to adopt a component-based architecture. In such an architecture, the policy is considered as loosely-coupled subpolicies. To build a complex global policy, they are integrated via various composition operators.

In recent years, research and development in policies were mainly around the two features mentioned above.

In a large system, there are many classes of subjects with different needs for processing a variety of resources. Different subjects usually have different (even competing) requirements on the use of resources and their security goals (confidentiality, availability, integrity) may be distinct. Hence, various access requirements have to be consistently authorized and maintained in a single policy. In this setting, the theory of security policy composition becomes crucially significant. The idea is similar to the component-based design in software engineering. That is, each simple and original module is firstly specified independently, then based on the control flow of the system or on policy requirements, the modules are composed into a whole system model. The objective is to deduce the properties of the whole system, based on the properties of the sub-modules, according to some theoretical results about property preservation (i.e., the overall policy preserves the properties of the constituent sub-policies).

A question that arises in composing policies is conflict resolution. The idea of disambiguating among possibly conflicting decisions appear in several works, such as in [1], [2] and is the core of the industrial standard access-control language XACML [3]. However, XACML combiners are in

This work was supported in part by National Natural Science Foundation of China with Grant No. 10701030 and by INRIA.

fact not sufficient when the decision conflicts cannot be solved based on the sub-policies decisions but on the activities of the subjects or the systems. For instance, in the context of preventing the conflict of interest between clients of competing companies, this is the case for the Chinese Wall policy, whose basic idea is that people are only allowed access to information which is not held to conflict with any other information that they already possess.

To address the safe composition problem, we introduce in this paper a systematic and formal methodology to model security policies and to verify whether required policy properties are preserved under composition of the sub-policies. The results presented in this paper are, up to the best of our knowledge, among the first efforts on systematic composition and analysis of security policies with Petri nets in the literature.

Petri nets are well-known for their graphical and analytical capabilities for the specification and verification of concurrent and distributed systems. Moreover, they have two main features particularly convenient for our methodology of modular security policy design.

1) Petri net representations are analytical and flexible. Analysis and logical reasoning can be performed on their representations and on their properties. They are compatible with a compositional approach via operators for compositions, refinements and reductions, and their functional purposes and characteristics are accurately and logically reflected.
2) Many Petri net-based techniques are available for verification, including reachability analysis or mathematical programming, as well as for characterization and transformations (see [4], [5] for a review.). There are also abundant results concerning property preserving operators.

Motivated by these advantages, several works about applying Petri nets to the policy design have appeared in the literature (Section 2 gives a brief review). The common characteristic of these approaches is that for a specific security policy, Petri nets are used for the specification, and the reachability-tree related techniques and CPN Tool are applied for the verification. However it is well-known that these techniques face a state explosion problem when the system is large and complex. In order to overcome this shortage and to strengthen the advantages of the Petri net formalism both in security policy and on software engineering areas, this paper presents some pioneer work about applying Petri nets for the security policy design specification and verification in a modular way. It provides the following contributions:

1) It introduces a newly defined model for the modular specification, i.e., Extended Petri net Process (EPNP), which are special Petri nets with a single entry place and a single exit place working as the module interfaces. In EPNP, colors are assigned to tokens and weights to distinguish different types of data and reduce the state space, time constraints are added to the transitions for specifying the duration of executing an operation.
2) It gives formal definitions for policy-related properties, namely, completeness, termination, consistency and confluence. These properties, presented in [6] in the rewriting framework, are here adapted to the Petri net approach and given in Petri net terminology. Some theoretical results concerning these properties are stated.
3) It specifies some policy composition operators based on Petri nets. Thanks to PPPA (a technology in Petri net theory applicable mainly for component-based system design in software engineering), eight simple composition operators based on EPNP are specified in order to give some hints on applying PPPA to the security policy design. The composition operators in PPPA are mainly useful for those policies that solves conflicts through system actions.
4) In order to solve conflicts in decisions obtained when policies are combined, according to XACML, it specifies four composition combiners to solve decision conflicts according to predefined rules.
5) For each composition operator, the preservation of policy properties is studied. This paper is the first one to specify security policy with EPNP and verify the policy properties based on the proof of property preservation, which is one of the most popular verification techniques in software engineering.

Our methodology of modeling and verification of security policy is highly flexible and scalable. It is flexible because any module (no matter whether or not it is obtained by composition of other sub-modules) can be safely replaced with an alternative design without reanalysis of the overall system architecture. Since each module is designed as a correct EPNP with a specific architecture, the replacement has the same interface and satisfies the same constraints. This feature is especially useful when we design different security policies with the same EPNP architectures (see the specification of the Bank COI in Chinese wall policy in Section 7.1 for example). It is scalable because it allows us to analyze overall composition without the interference of internal details of the module design. Verification is done separately by checking whether each sub-module satisfies the constraints of property preservation. This significantly reduces the complexity. Furthermore, our methodology is general and can be applied to a large range of security policies design.

The paper is organized as follows: after giving some related works in Section 2, Section 3 reminds some basic terminology about Petri nets and the definition of extended Petri net process; Section 4 provides the formal definition of policy properties with Petri net terminology, and some related results concerning the policy properties are given. Section 5 is about the modular security policy design technology with a property preserving approach. Eight composition operators based on EPNP are defined and studied. Then in Section 6, XACML combiners are specified with EPNP. For each combiner, property preservation results are presented. In Sections 7 a large scale complex policy design is illustrated based on the methodology mentioned in the previous sections. Some conclusive remarks are given in Section 8.

2 Related Work

While there is a huge literature in systems engineering on the key idea that complex systems are built by assembling components, considering security policies as components and studying their composition is a relatively recent research trend.

Policy composition is addressed in [7] through an algebra of composition operators that is able to define union, intersection, policy templates, among other operations. The work presented in [8] extended this algebra with negative authorizations and non-determinism and also includes an operator for sequential composition. Another alternative for composing access control policies is implemented by the Polymer system [9]. A different approach to composition is taken in [10] for composing policy specifications for web-services security. In [11], the author proposes a set of high-level composition operators coherent with a four-valued logic for policies.

In recent years, term rewriting theory has been applied for the specification and verification of security policy design [6], [12], [13] and composition of security policy is addressed in [6], [14]. In a rewrite-based specification, policies are expressed by rules close to natural language: if some conditions are satisfied, then, a given request is evaluated into a decision, for instance it may be granted or denied. This expressivity allows one to finely specify the conditions under which decision takes place and these conditions may involve attributes related to subjects or resources. Moreover strategic rewriting is used to express control on the rules and to handle priorities or choices between possible decisions. For instance the specification of XACML combiners is given in [6], [14] in the rewriting context.

Using Petri nets for the specification and verification of the security policy design is not a new story. In [15], a colored Petri-net based framework is presented for verifying the consistency of RBAC policies. The reachability analysis technique is applied for RBAC policy verification. In [16], CPN is used to specify a real industrial example, namely an access control system developed by the Danish security company Dalcotech A/S. Based on the CPN model, the Design/CPN tool is applied for the implementation of automatic code generation. [17] defines task based access control as a dynamic workflow and then specifies the workflow with Petri nets. [18] and [19] model Chinese wall policy and Strict Integrity Policy, respectively, with CPN and apply coverability graph for the verification. [20] uses CPN for the specification of mandatory access control policies and occurrence graph is applied for verification. [21] applies Predicate/Transition net for the modeling and analysis of software security system architectures.

3 Extended Petri Net Process

In this paper, we assume some familiarity with basic notions on Petri nets, briefly introduced here. More terminology and fundamentals on Petri net theory can be found in [4], [22].

Definition 1 (Petri nets): A Petri net (N, M_0) is a net $N = (P, T, F, W)$ with an initial marking M_0 where,

- P is a finite set of places of cardinality $|P|$;
- T is a finite set of transitions such that $P \cap T = \phi$, and $P \cup T \neq \phi$;
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation;
- W is a weight function such that $W(x,y) \in \mathcal{N}^+$ if $(x,y) \in F$ and $W(x,y) = 0$ if $(x,y) \notin F$. For any $X, Y \subseteq P \cup T$, we denote $W(X,Y) = \{W(x,y) \mid x \in X, y \in Y, (x,y) \in F\}$;
- M_0 is a function $M : P \to N$ such that $M(p)$ represents the number of tokens in place $p \in P$.

Definition 2 (firing rule): A transition $t \in T$ is firable (or enabled) at a marking M if and only if $\forall p \in P : (M(p) \geqslant W(p,t))$. Firing (or executing) transition t results in changing marking M to marking M', where $\forall p \in P : (M'(p) = M(p) - W(p,t) + W(t,p))$.

Definition 3 (pre-set, post-set, input set and output set): For $x \in P \cup T$, ${}^\bullet x = \{y | (y,x) \in F\}$ and $x^\bullet = \{y | (x,y) \in F\}$ are called the pre-set (input set) and post-set (output set) of x, respectively. For a set $X \subseteq P \cup T$, ${}^\bullet X = \cup_{x \in X} {}^\bullet x$ and $X^\bullet = \cup_{x \in X} x^\bullet$.

Definition 4 (incidence matrix): The pre-incidence matrix PRE of a net N is a $|P| \times |T|$ matrix whose element at row p and column t is the weight $W(p,t)$ of the arc from place p to transition t. The post-incidence matrix POST of N is a $|P| \times |T|$ matrix whose element is the weight $W(t,p)$ of the arc from transition t to place p. $V = POST - PRE$ is called the incidence matrix of N.

Definition 5 (state equation and firing count vector): For a Petri net (N, M_0), $M = M_0 + V\mu$, is called the state equation, where V is the incidence matrix of N and $\mu \in N^T$ is the firing count vector of a firing sequence σ, i.e., $\mu[t]$ is the number of times transition t occurs in σ .

Any reachable marking M satisfies the state equation [22]. In other words, if the state equation is not true, then the corresponding marking M is not reachable.

Definition 6 (Extended Petri nets): An extended Petri net $EPN = (N, M_0, C, \tau)$ is a Petri net with a color set C assigned to the tokens in the places and the weight on the arc and a time set τ assigned to the transitions.

Usually, colors in the EPN are used to distinguish different types of data while time constraints are used to specify the duration of executing an operation. In an EPN, the marking and the weight are denoted as multiple dimension vectors. A transition $t \in T$ is firable (or enabled) at a marking M if and only if $\forall p \in P : M(p) \geq W(p,t)$, where both $M(p)$ and $W(p,t)$ are $|C|$-dimensional vectors. Firing (or executing) transition t results in changing marking M to marking M', where $\forall p \in P : (M'(p) = M(p) - W(p,t) + W(t,p))$.

For denoting a marking of EPN, we have different expressions that may be used interchangeably throughout the paper:

- a vector expression: e.g., for an EPNP with two colors c_1, c_2, a marking with three places, for instance $M = ((2,1),(0,0),(1,1))$, meaning that there are two tokens with color c_1, one token with color c_2 in place p_1, no tokens in place p_2 and two tokens with color c_1 and c_2 respectively in place p_3;

- a multi-set expression: e.g., for the above marking, we denote $M = \{2c_1, c_2\}p_1 + \{c_1, c_2\}p_3$. Correspondingly, sometimes we use just a color set to denote the marking in a place. e.g., $M(p_1) = \{2c_1, c_2\}$, and the number of tokens in place p_1 is $|M(p_1)| = 3$;
- the mix of vector and multi-set expression; e.g., for the above marking M and a new place p_4 with one token, we may use $M + p_4$ to denote another marking which includes four places.

Definition 7 (firing sequence and reachability): Let M, M' be markings, t be a transition, and σ be a transition sequence in a Petri net (N, M_0). $M[N, \sigma\rangle M'$ means that M' is reachable from M by firing σ. $M[N, *\rangle M'$ means that M' is reachable from M by firing an unspecified sequence. $R(N, M)$ denotes the reachability set of N starting from M, i.e., the smallest set of markings such that: (a) $M \in R(N, M)$; (b) If $M' \in R(N, M)$ and $M'[N, t\rangle M''$ for some $t \in T$, then $M'' \in R(N, M)$.

For example, in the EPN shown in Fig.1, initially, there are two types of colored tokens $< 1 >$ and $< 2 >$ in place p_1, the initial marking is $M_0 = ((2, 2), (0, 0), (0, 0), (0, 0))$ (or $M_0 = ((2, 2), 0, 0, 0)$ for simplicity). Both t_1 and t_2 are firable, after firing t_1 for 3 units of time, a reachable marking $M_1 = ((1, 1), (1, 0), 0, 0)$.

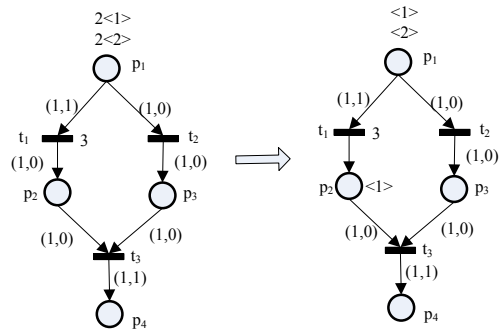

Fig. 1. An extended Petri net example.

Definition 8 (Extended Petri Net Process (Fig.7)): An extended Petri Net Process (EPNP) $B = (EPN, p_e, p_x)$ is an extended Petri net EPN with an additional unique entry place p_e and unique exit place p_x, where the place p_e (resp. p_x) has no input (resp. output) transitions.

When the security policy is specified with an EPNP, the entry place p_e represents the request, while the exit place p_x represents the decision. Different requests and different decisions are distinguished with different colored tokens. At the same time, the entry place and exit place are designed as two interfaces of the process. The initial marking of an EPNP is denoted as $M_e = M_0 + p_e$, the exit marking is $M_x = M + p_x$, where M is a marking in the internal EPN.

In the remaining part of this section, let us emphasize and discuss several features of Extended Petri Net Processes.

1) Uniqueness of the entry place and exit place: by definition, an EPNP has only one entry place and one exit place. This uniqueness assumption is mainly for the consistency in modeling and convenience in creating composite processes under the various operators. For modeling real-life problems, the case of multiple entries (resp., exits) can be easily converted to the case of single entry (resp., exit) by creating a super entry (resp., exit) place and controlling the firing of its output (resp., input) places.
2) Role of the exit place p_x: in a Petri net model like EPNP, p_x is simply a sink place, indicating the location where the control flow may leave the process after one cycle of executions.
3) M_0 represents a token distribution assigned to the set of places P before execution of the EPNP B starts. Those places having tokens serve controlling purposes. For example, they may represent some system resources that are available before B starts its execution. The association of a static marking M_0 with B, where $M_0 \neq 0$, is a special feature of EPNP. It greatly enlarges the scope of application of EPNP.
4) Proper Initiation: it is not guaranteed that (B, M_e) can always be initiated. However, if B can ever start firing, it must start at M_e and not at any other marking.
5) Deadness of Static Marking: together with the previous condition, this feature implies that given a static marking, a process can only be initiated as follows:
   - A process can start execution only after some tokens have been deposited into its entry place p_e.
   - Without this deposit of tokens, (B, M_0) cannot 'self-start'. This reflects the realistic requirement that a process cannot start by itself. In order to start, it must be called by another process or by itself (i.e. recursively).

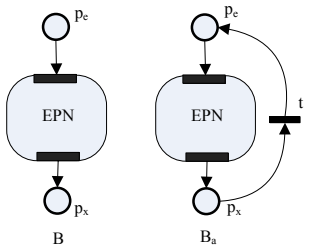

Fig. 2. An EPNP and its associated net.

Since an EPNP is not strongly connected, it cannot satisfy those important system properties such as liveness, reversibility and so on. However, they can be recovered by considering the associated net of an EPNP B, that is a net with an additional transition t and two arcs (p_x, t) and (t, p_e) in B (Fig. 2). An EPNP is called almost live (respectively, bound, reversible, etc) if its associated net is live (respectively, bound, reversible, etc).

4 Petri Net Based Properties For Security Policy

In our approach, security policies are build in a modular way from basic modules, specified with extended Petri net

processes. Accordingly, the policy properties are now defined on EPNP.

4.1 Completeness

A security policy is decision complete (or simply complete) if it computes at least one decision for every incoming request. This property is called totality in [23] and [24].

Definition 9 (Completeness): Suppose a security policy is specified with an EPNP $B = (EPN, p_e, p_x)$. The policy B is complete if for any initial marking M_e, there exists a marking $M_x = M + p_x$ which is reachable from M_e.

Based on the definition, an initial marking represents a request, and the exit marking $M + p_x$ is reachable, implying that the policy will return a decision.

4.2 Termination

A security policy terminates if the evaluation of every incoming request terminates.

Definition 10 (Termination): Suppose a security policy is specified with an EPNP $B = (EPN, p_e, p_x)$. The policy B is terminating (or strongly terminating) iff B has no infinite firing sequences for any initial marking. B is said weakly terminating iff for any initial marking, B has at least one finite firing sequence. If B terminates with an exit marking $M_x = M_0 + p_x$, B is called properly terminating.

If B is weakly terminating, it may have infinite firing sequence(s) but must terminate in some cases (see Fig. 3). Strong termination requires that the policy always terminates with a finite number of firing steps; while properly terminating requires the policy to terminate (strongly or weakly) and to reach a special exit state. More discussions on proper termination are as follows:

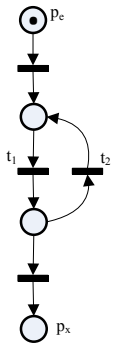

Fig. 3. An example of a weakly terminating policy.

- In general, proper termination by itself does not guarantee that a process can always terminate. It just requires a process to be at the exit state $M_x = M_0 + p_x$ whenever a token has been deposited into p_x. Proper termination models the well-known 'memoryless' property of a software process that it should return to its initial 'ready' state after having completed a cycle of execution.
- Together with the Deadness of Static Marking condition, proper termination guarantees that no transition can be fired when p_x gets a token. This follows from the fact, whenever p_x gets a token, the system reaches a dead marking because M_0 in the internal EPN is dead.
- The Deadness of Static Marking condition, together with the properties of Proper Initiation and proper termination, guarantee that an EPNP is non-reenterable. This means that, once having been initiated, an EPNP cannot be initiated again until its previous execution cycle has been completed. In general, to avoid mixing two independent execution cycles of a Petri net process, one either has to use colored Petri nets or control the procedure of entering into the process.

The following results allow us to connect these policy properties and the usual notions of boundedness, reversibility, liveness and deadlock-freeness in Petri nets [22].

Proposition 1: Suppose a security policy is specified with an EPNP $B = (EPN, p_e, p_x)$. The following properties are true:

1) If B is almost live, B is complete;
2) If B terminates properly, B is complete;
3) If B can terminate from any reachable marking and is deadlock-free, then B is complete;
4) If B is complete and almost bounded, then it is (strongly or weakly) terminating.
5) If B is almost live and bounded, then it is terminating.
6) If B is almost live and reversible, then it is properly terminating.

Proof:

1) If B is almost live, then for any initial marking M_e specifying a request, there exists a reachable marking such that the transition $t \in {}^\bullet p_x$ is firable and $M_x = M + p_x$ is reached after firing the transition t.
2) If B terminates properly, then $M_x = M_0 + p_x$ is a reachable marking and hence B is complete.
3) By contradiction. If B is not complete, then there exists a reachable marking such that either B cannot terminate or it reaches a dead marking. This is in contradiction with the assumption.
4) By contradiction. If B cannot terminate, then either there is a cycle or B has an infinite firing sequence. As a result, either the tokens keep on transferring in the cycle and B cannot be complete or the associated net of B is unbounded. This is in contradiction with the assumption.
5) If B is almost live, then B is complete (based on property 1), and by property 4, B is terminating.
6) Since B is almost live, by property 1, B is complete. That is, $M_x = M + p_x$ is reachable from $M_e = M_0 + p_e$. Since B is almost reversible, $M_0 + p_x$ is reachable from $M + p_x$ in the associated net B_a. Hence $M_x = M + p_x = M_0 + p_x$ in B and B is properly terminating.

□

4.3 Consistency

A security policy is consistent if it computes at most one access decision for any given input request.

Definition 11 (Consistency): Suppose a security policy is specified with an EPNP $B = (EPN, p_e, p_x)$. Then the policy B is consistent iff for any initial marking $M_e = M_0 + p_e$, all reachable markings M_i satisfy that $|M_i(p_x)| \leq 1$ and for any exit markings M_j and M_k, $M_j(p_x) = M_k(p_x)$.

Consistency implies that for any request, the policy returns at most one decision. According to the above definition, all reachable markings can have at most one token ($|M_i(p_x)| \leq 1$) with a unique identical color in place p_x (since $M_j(p_x) = M_k(p_x)$). Note that when the EPNP does not terminate, the decision place p_x will not be marked, so the consistency property is trivially satisfied.

The consistency property can be related to state equations in Petri net theory. In a general Petri net (N, M_0), any reachable marking M satisfies the state equation $M = M_0 + V\mu$, where V is the incidence matrix and μ is the count vector of a firing sequence σ and $M_0[N, \sigma\rangle M$ (see Definition 5).

Proposition 2: Suppose a security policy is specified with an EPNP B. Then, B is consistent if

- at most one of the following state equations is satisfied: $M_i = M_e + V\mu_i$, where $|M_i(p_x)| = 1$
- and no one of the following state equations is satisfied: $M_i = M_e + V\mu_i$, where $|M_i(p_x)| > 1$.

Proof: If the first state equations cannot be satisfied, then the policy cannot make a decision. If at most one of first state equations is satisfied, this implies that there may exist a decision. If the second equation cannot be satisfied, this implies that the case of more than two different decisions is impossible. □

Let us now relate consistency and the notion of confluence defined for Petri nets.

4.4 Confluence

Definition 12 (Confluence): Suppose a security policy is specified with an EPNP $B = (EPN, p_e, p_x)$. The policy B is confluent iff for any initial marking $M_e = M_0 + p_e$ and any two reachable markings $M_i, M_j \in R(B, M_e)$, there exists a reachable marking M_c in B such that $M_c \in R(B, M_i) \cap R(B, M_j)$.

A home space of B, denoted HS, is a set of markings, such that for any $M_i, M_j \in R(B, M_e)$, there exists at least one marking M_c in HS reachable from both M_i and M_j. If a HS contains only one element M_c, then M_c is called a home marking of B. In other words, a home marking is reachable from any marking $M \in R(B, M_e)$.

The confluence property has been studied in the literature [25], [26], [27], [28]. It is proved to be a decidable property in Petri net theory. For ordinary Petri nets (with weight 1 on each arc and no self-loop), the confluence can be reduced to confluence of a 2-shallow term rewriting systems [27]. The following Proposition is extracted from [27], [28] and the detailed proof can be found in [28].

Proposition 3: Suppose a security policy is specified with an EPNP B.

1) If a Petri net has a home marking then it is confluent.
2) A safe Petri net (i.e., a PN which satisfies that the number of tokens in any place cannot exceed one for any reachable marking) has a home marking iff it is confluent.
3) Any confluent and strongly terminating Petri net has a unique home marking.

Proposition 4: Suppose a security policy is specified with an EPNP B. If B is consistent and proper terminating, then B is confluent and has a unique home marking.

Proof: Since B is proper terminating and consistent, for any request marking $M_e = M_0 + p_e$ there exists a unique exit marking $M_x = M_0 + p_x$ reachable from M_e. From any two reachable markings M_i and M_j, one can reach M_x. Hence B is confluent and M_x is the unique home marking. □

5 Modular Policy Composition Based on EPNP

In this section, our focus is on the composition of the security policies in a modular way. In general, combining security policies may result in inconsistent or non-terminating policies. We explore which syntactic conditions and which operators can guarantee the preservation of these suitable properties for the composition of two policies.

PPPA is the abbreviation of Property preserving Petri net Process Algebra [4], [5]. The algebra defines about twenty operators based on PNP and considers the preservation of about twenty system properties. The details of the formal definition of these operators and the proof of the property preservation results can be found in two PhD theses [4], [5].

In order to give some hints about how to apply PPPA to the specification of security policies, we restrict our attention in this paper to only four logic related composition operators, namely Enable, Choice, Interleave and Disable, and four application related compositions, namely place merging, transition merging, place refinement and transition refinement, and to the security-policy related properties, i.e., completeness, termination, consistency and confluence. The application of other operators follows similar ideas.

5.1 Logic based Operators for Composition

Definition 13 (Enable (Fig. 4)): For two processes $B_i = (P_i, T_i, F_i, W_i, M_{i0}, C_i, P_{ie}, P_{ix})$ $(i = 1, 2)$, their composition by Enable, denoted $B_1 >> B_2$, is defined as the process $B = (P, T, F, W, M_0, C, p_e, p_x)$, where $P = P_1 \cup P_2$, $p_e = p_{1e}, p_x = p_{2x}$ and p_{2e} is merged with p_{1x}; $T = T_1 \cup T_2; F = F_1 \cup F_2$; $W = W_1 \cup W_2$; $M_0 = M_{10} \cup M_{20}$; $C = C_1 \cup C_2$.

The Enable composition $B_1 >> B_2$ models the sequential execution of two processes B_1 and B_2. That is, B_1 is first executed and B_2 is executed after the successful termination of B_1. However, if B_1 does not exit successfully, B_2 will never be activated.

Proposition 5: Let B be the policy obtained from two sub-policies B_1 and B_2 by applying the composition operator Enable. Then,

1) B is complete iff B_1 and B_2 are complete.

2) B is strongly (resp., properly) terminating iff B_1 and B_2 are strongly (resp., properly) terminating; B is weakly terminating iff B_1 and B_2 are weakly terminating, or B_1 is complete and B_2 is weakly terminating.
3) If both B_1 and B_2 are consistent, then B is consistent.
4) If both B_1 and B_2 are confluent, then B is confluent.

Proof: For each firing sequence σ in B, it is either a firing sequence in B_1 or a union of a sequence σ_1 in B_1 and a sequence σ_2 in B_2, where $M_{1e}[B_1, \sigma_1\rangle M_{1x}$ in B_1. The remaining part of the proof is trivial. □

Definition 14 (Choice (Fig. 4)): For two processes $B_i = (P_i, T_i, F_i, W_i, M_{i0}, C_i, p_{ie}, p_{ix})$ $(i = 1, 2)$, their composition by Choice, denoted $B_1[]B_2$, is defined as the process $B = (P, T, F, W, M_0, C, p_e, p_x)$, where $P = P_1 \cup P_2$, p_e is the place merging p_{1e} and p_{2e}, p_x is the place merging p_{1x} and p_{2x}; $T = T_1 \cup T_2$; $F = F_1 \cup F_2$; $W = W_1 \cup W_2$; $M_0 = M_{10} \cup M_{20}$; $C = C_1 \cup C_2$.

The Choice composition $B_1[]B_2$ models the arbitrary selection for execution between two processes B_1 and B_2.

Proposition 6: Let the policy B be obtained from two sub-policies B_1 and B_2 by applying the composition operator Choice. Then,

1) B is complete iff B_1 and B_2 are complete;
2) B is strongly (resp., weakly, properly) terminating iff B_1 and B_2 are strongly (resp., weakly, properly) terminating;
3) B is consistent if B_1 and B_2 are consistent and output the same colored token in the exit places;
4) B is not always confluent even if both B_1 and B_2 are confluent. B is confluent if B_1 and B_2 are proper terminating and output the same token in their exit place.

Proof: After applying Choice operator, the control flow is within one of the sub-policies, so the first two properties are trivial. For property 3, although both B_1 and B_2 are consistent, they may output different decisions. Hence B is not always consistent unless both B_1 and B_2 always output the same colored token in their exit place. As for property 4, suppose $M_i \in R(B_i, M_{ie}), i = 1, 2$. Then $M_i \in R(B, M_e)$. There is no reachable marking $M \in R(B, M_1) \cap R(B, M_2)$ except $M = M_x = M_0 + p_x$. Hence, B is not confluent unless both B_1 and B_2 terminate properly and output the same colored token in their exit place. □

Definition 15 (Interleave (Fig. 5)): For two processes $B_i = (P_i, T_i, F_i, W_i, M_{i0}, C_i, P_{ie}, P_{ix})$ $(i = 1, 2)$, their composition by Interleave, denoted $B_1|||B_2$, is defined as the process $B = (P, T, F, W, M_0, C, p_e, p_x)$, where $P = P_1 \cup P_2 \cup \{p_{1e}, p_{2e}\}$, p_e, p_x are the newly added entry place and exit place, respectively; $T = T_1 \cup T_2 \cup \{t_0, t_c\}$, where t_0, t_c are newly added transitions; $F = F_1 \cup F_2 \cup \{(p_e, t_0), (t_0, p_{1e}), (t_0, p_{2e}), (p_{1x}, t_c), (p_{2x}, t_c), (t_c, p_x)\}$; $W = W_1 \cup W_2 \cup \{W((p_e, t_0), W(t_0, p_{1e}), W(t_0, p_{2e}), W(p_{1x}, t_c), W(p_{2x}, t_c), W(t_c, p_x)\}$, where $W(t_c, p_x)$ is a 2-dimension vector $(W(p_{1x}, t_c), W(p_{2x}, t_c))$; $M_0 = M_{10} \cup M_{20}$; $C = C_1 \cup C_2$.

The Interleave composition $B_1|||B_2$ models the concurrent but independent execution of two processes B_1 and B_2 with synchronized exit.

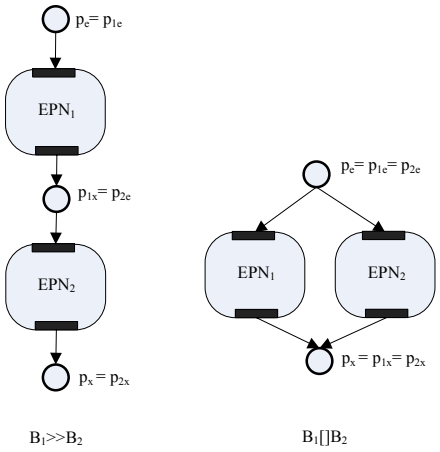

Fig. 4. The operators Enable and Choice.

Proposition 7: Let the policy B be obtained from two sub-policies B_1 and B_2 by applying the composition operator Interleave. Then,

1) B is complete iff B_1 and B_2 are complete;
2) B is strongly (resp., properly) terminating iff B_1 and B_2 are strongly (resp., properly) terminating; B is weakly terminating iff B_1 and B_2 are weakly terminating;
3) B is consistent iff B_1 and B_2 are consistent;
4) B is confluent iff B_1 and B_2 are confluent.

Proof:

1) B is complete iff transition t_c is firable, i.e., both B_1 and B_2 are complete.
2) Since B_1 and B_2 are executed independently, each firing sequence of B is a union of sequences of B_1 and B_2. Property 2 follows easily.
3) If both B_1 and B_2 are consistent, for a request, the outputs of each sub-policy are always the same, by Definition 15, the output of transition t_c is unique and B is consistent. On the other hand, if B is consistent, the token color (a 2-dimension vector) in p_x is unique, correspondingly, each entry of the 2-dimension vector is unique, i.e., the token color in places p_{1x} and p_{2x} should be unique, implying both B_1 and B_2 are consistent.
4) For any two reachable markings $M_i = P_{im} + Q_{im}$ in B, where P_{im}, Q_{im} are markings in B_1 and B_2 respectively and $i = 1, 2$, since both B_1 and B_2 are confluent, there exist $M'_1 \in R(B_1, P_{1m}) \cap R(B_1, P_{2m})$ and $M'_2 \in R(B_2, Q_{1m}) \cap R(B_2, Q_{2m})$. Then $M = M'_1 + M'_2 \in R(B, M_1) \cap R(B, M_2)$ and B is confluent.

□

Definition 16 (Disable (Fig. 6)): For two processes $B_i = (P_i, T_i, F_i, W_i, M_{i0}, C_i, P_{ie}, P_{ix})$ $(i = 1, 2)$, their composition by Disable, denoted $B_1[> B_2$, is defined as the process $B = (P, T, F, W, M_0, C, p_e, p_x)$, where $P = P_1 \cup P_2 \cup \{p_{1e}, p_{2e}\}$, p_e, p_x are the newly added entry place and exit place, respectively; $T = T_1 \cup T_2 \cup \{t_0, t_c\}$; $F = F_1 \cup F_2 \cup \{(p_e, t_0), (t_0, p_{1e}), (t_0, p_{2e}), (p_{1x}, t_c), (p_{2x}, t_c), (t_c, p_x)\} \cup \{(P_d, T_d), (T_d, p_{2x})\}$, where $P_d \subseteq P_2, T_d \subseteq T_1$; $W = W_1 \cup W_2 \cup$

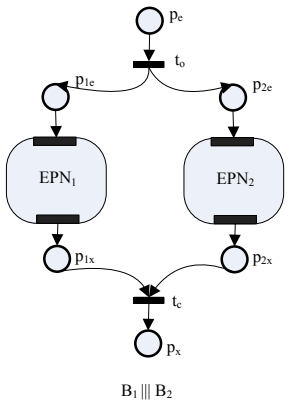

Fig. 5. The opertor Interleave.

$\{W(p_e, t_0), W(t_0, p_{1e}), W(t_0, p_{2e}), W(p_{1x}, t_c), W(p_{2x}, t_c), W(t_c, p_x)\}$ $\cup \{W(P_d, T_d), W(T_d, p_{2x})\}$, where $W(t_c, p_x)$ is a 2-dimension vector $(W(p_{1x}, t_c), W(p_{2x}, t_c))$; $M_0 = M_{10} \cup M_{20}$; $C = C_1 \cup C_2$.

The Disable composition is similar to Interleave. The difference is that there exist some places P_d in B_2 which are connected to some transitions T_d in B_1 such that once T_d are fired, B_2 is dead and cannot output decisions normally. According to the policy requirement, we may add an additional arc (T_d, p_{2x}) for specifying the decisions of sub-policy B_2 if it is disabled.

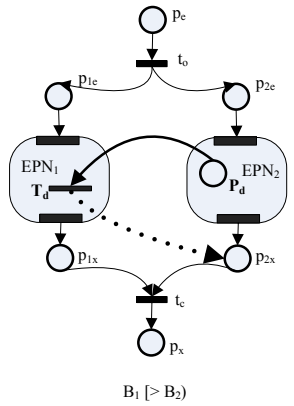

Fig. 6. The operator Disable.

Proposition 8: Let the policy B be induced from two sub-policies B_1 and B_2 by applying the composition operator Disable. Then,

1) Suppose the transitions in T_d are never fired in B. Then the property preservation results are the same as those for the Interleave operator in Proposition 7.
2) Suppose some transitions in T_d are fired and all the transitions in B_2 are disabled in B. Then,
   - B is complete if B_1 is complete;
   - B is strongly terminating if B_1 is strongly terminating; B is weakly terminating if B_1 is weakly terminating;
   - B is consistent if B_1 is consistent;
   - B is confluent if B_1 is confluent.
3) If B_1 and B_2 are both complete (resp., terminating), B is complete (resp., terminating), but in general confluence and consistency are not preserved.
4) Suppose B_2 satisfies the following conditions: $^\bullet(P_d^\bullet) = \{p_d\}$, $P_d^\bullet = {}^\bullet p_{2x}$, and $W(P_d, {}^\bullet p_{2x}) = W(P_d, T_d), W({}^\bullet p_{2x}, p_{2x}) = W(T_d, p_{2x})$. Then B is consistent (resp., confluent) iff B_1 and B_2 are consistent (resp., confluent).

Proof:

1) If T_d are never fired in B, the operator is the same as Interleave.
2) Once T_d are fired, B_2 is dead and the remaining flow is occurring in B_1. Hence, B preserves the properties of B_1.
3) Since both B_1 and B_2 are complete, after applying the Disable operator, transition t_c is firable and hence B is complete. Generally, T_d may be fired or not. But in both cases, the length of firing sequences in B cannot exceed the sum length of two firing sequences selected from B_1 and B_2, respectively. Hence, B preserves the termination property. On the contrary, the consistency and confluence properties cannot be preserved in general: since T_d may be fired, B may produce a decision which is different from a decision of B_1. In that case B is obviously not consistent. For a reachable marking M in B, suppose T_d is firable and $M[B, T_d\rangle(P_{1m} + Q_{1m})$ and $M[B, *\rangle(P_{2m} + Q_{2m})$, where Q_{1m} is a dead marking in B_2, while Q_{2m} is a reachable marking in B_2. Then, there does not exist a reachable marking that belongs to $R(P_{1m} + Q_{1m}) \cap R(P_{2m} + Q_{2m})$. Hence, B is not confluent in general.
4) When B_2 satisfies the given conditions, the reachable marking (i.e. the final decision) in B_2, resulted from firing transitions in $P_d^\bullet$, is the same as that resulted from firing T_d. Hence, whether or not T_d is fired, the reachable marking states in B_2 are the same. The remaining part of the proof is similar to the proof for Proposition 7 and omitted.

□

5.2 Policy Composition via Resources Sharing

In the context of cooperation between several independent systems or entities, two or more security components have to interoperate for accessing resources from the cooperating domains. In this case, a global policy should be designed by composing the local policies for sharing resources.

Example 1 (printer accessing policy (PAP)): Given two domains D_1 and D_2, let us assume that each domain has some resources (printers and xeroxing machines) for use. The local policy for accessing resources is that once the requested resources are available, the local user can access them; after being used, the resources should be released. For a global domain D composed from D_1 and D_2, the resources access policy is not changed, that is, each user can access any resource once it is available and must release the resource

after using. The difference of global policy and local policy is that the set of resources is changed.

For EPNP specification, the policy composition is formally defined as follows:

Definition 17 (place fusion (Fig. 7)): For two processes $B_i = (P_i \cup R, T_i, F_i, W_i, M_{i0}, C_i, \tau_i, p_{ie}, p_{ix})$ $(i = 1, 2)$, their composition by sharing resources R, where, $M_{10}(R) = M_{20}(R)$, denoted $B_1[R]B_2$, is defined as the EPNP $B = (P \cup R, T, F, W, M_0, C, \tau, p_e, p_x)$, where $P = P_1 \cup P_2 \cup \{p_{1e}, p_{2e}, p_{1x}, p_{2x}\}$, p_e and p_x are newly added interface places; $T = T_1 \cup T_2 \cup \{t_e, t_x\}$; $F = F_1 \cup F_2 \cup \{(p_e, t_e), (t_e, p_{1e}), (t_e, p_{2e}), (p_{1x}, t_x), (p_{2x}, t_x), (t_x, p_x)\}$; $W = W_1 \cup W_2 \cup \{W(p_e, t_e), W(t_e, p_{1e}), W(t_e, p_{2e})\} \cup \{W(p_{1x}, t_x), W(p_{2x}, t_x), W(t_x, p_x)\}$; where $W(t_x, p_x)$ is a 2-dimension vector $(W(p_{1x}, t_x), W(p_{2x}, t_x))$ $M_0 = M_{10} \cup M_{20}$; $C = C_1 \cup C_2$; $\tau = \tau_1 \cup \tau_2$ $(\tau(t_e) = \tau(t_x) = 0)$.

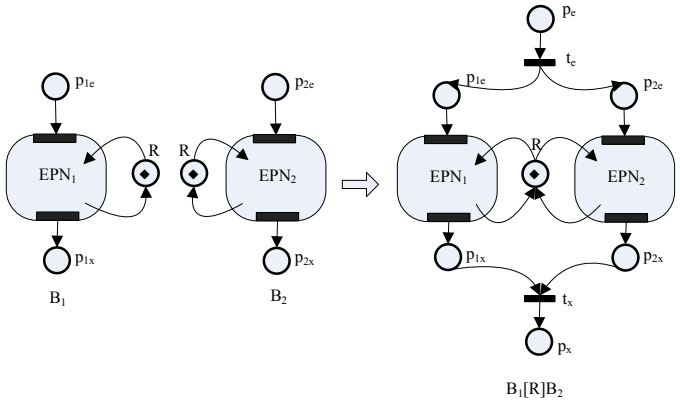

Fig. 7. Policy composition via resource sharing

Based on the definition, place fusion operator is applicable for two EPNPs with the same set of resource places for sharing, i.e., they have the common resource places R with the same number of initially marked tokens. Hence, for policy composition, the EPNP specification for each local policy may need a modification before applying fusion operator. That is, adding some new resource places (e.g., place r_2 in Fig. 8(a) is newly added) and modify the number of tokens for some resource places (e.g., the number of tokens in r_1 and r_2 in Fig. 8(b) are changed).

In order to illustrate the above definition, let us consider Example 1. In each local domain, a user may request printing, or copying, or printing and copying, or doing nothing. Based on local policy, once the resource is available, the decision for the request is permitted. Let us suppose that the domain D_1 has one printer (r_1) and no xeroxing machine, the domain D_2 has one printer (r_1) and one xeroxing machine (r_2). The EPNP based policy specification is shown in Fig. 8(a), where a user from D_1 can only successfully request printing (firing t_{51}) or doing nothing (firing t_{91}). Neither copying nor "printing and copying" is possible. In domain D_2, a user can successfully request printing (firing t_{52}), or copying (firing t_{72}), or printing and copying (firing t_{12}), or doing nothing (firing t_{92}) once the requested resource is available.

In this cooperation context, the global domain now has two printers and one xeroxing machine for sharing. Hence, Fig. 8(a) will be changed to Fig. 8(b) by modifying their resource places r_1 and r_2. At last, place fusion operator is applied for policy composition and resulted in Fig. 8(c). The specification of the places and transitions are explained in Table 1.

Note that resource sharing may result in unsafe interoperation. For instance, in the above example, copying is not permitted in D_1 (transition t_{71} does not appear in Fig. 8(a)) but is permitted in the global domain (transition t_{71} may be enabled in Fig. 8(c)). However, whether or not to cooperate is the manager's business and it is out the scope of this paper. Instead, this paper focuses on how to compose the policies and assumes that adding new resources to a local policy is safe. For example, in Example 1, the manager decides whether or not to share resources (i.e., to transform Fig. 8(a) into Fig. 8(b)), while our business is to consider how to specify sharing (i.e., to transform from Fig. 8(b) into Fig. 8(c)).

TABLE 1
Specification of Fig. 8

node	specification
p_e, p_x	the interface places of global policy
p_{ie}, p_{ix}	the interface places of local policies
p_{1i}	The state of printing and copying
p_{2i}	The state of finishing printing
p_{3i}	The state of finishing copying
p_{4i}	The state of printing
p_{5i}	The state of copying
r_1	The resource place for printers
r_2	The resource place for xeroxing machine
t_e, t_x	The interface transitions of the global policy
t_{1i}	request printers and xeroxing machines
t_{2i}	printing and copying (with time constraint)
t_{3i}, t_{6i}	release printers and copying (with time constraint)
t_{4i}, t_{8i}	release xeroxing machines
t_{5i}	request printers and printing (with time constraint)
t_{7i}	request xeroxing machines and copying (with time constraint)
t_{9i}	request nothing

Generally, composing two systems by place fusion may result in system deadlocks [29], [30]. Deadlock occurs when resources are limited and users should compete for using them.

For handling deadlock issues based on Petri net models, current research considers the following three types of approaches: the first one is relying on techniques concerning siphons and traps. Elementary siphon invariants in Petri net structures are useful for analyzing deadlock [31], [32], [33], [34]. The deadlock-free method is addressed by adding a monitor or controller to avoid deadlock structure [33], [34], [29]. The second approach relies on a scheduling algorithm. Heuristic scheduling algorithms, such as genetic algorithms, are used to get an optimum and deadlock-free scheduling of flexible manufacturing systems in [35], [36]. The third approach is based on the notion of transitive matrix [37]. The place transitive matrix describes the transferring relation from one place to another place through transitions. The analysis of the cyclic scheduling for the determination of the optimal cycle time is studied, and the concept of transitive matrix

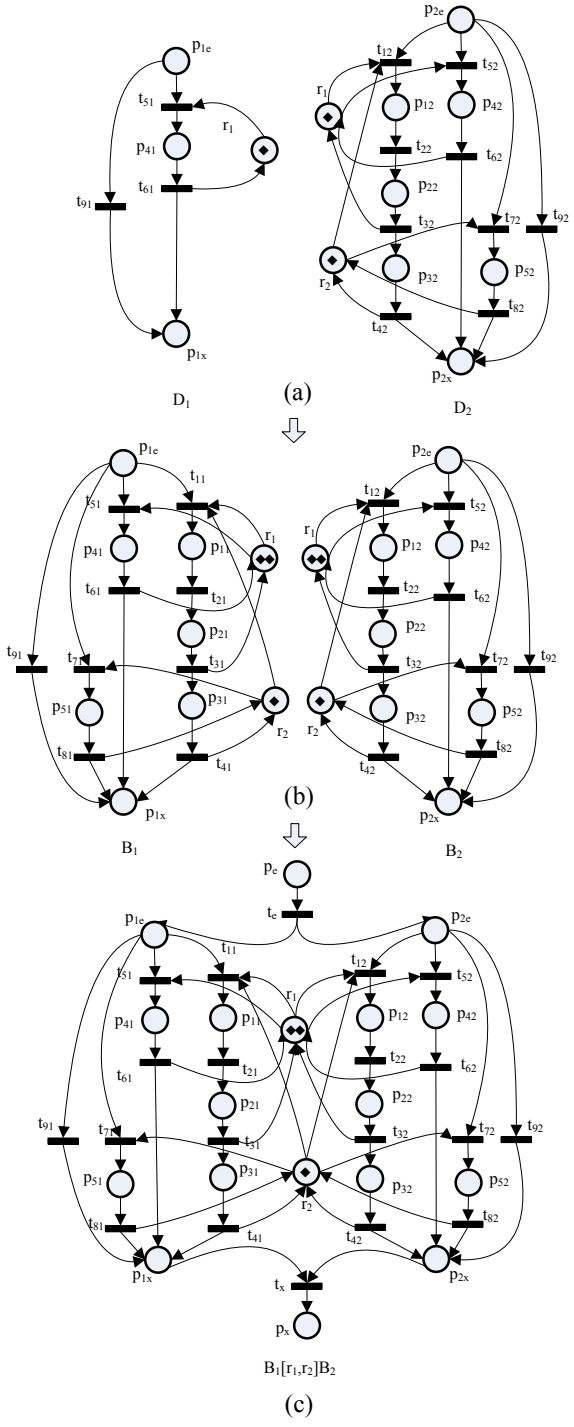

Fig. 8. EPNP-based specification for Example 1

has efficiently been used to slice off some subnets from the original net in [30]. Deadlock-free conditions are given in [38], [39] based on transitive matrix, and an algorithm for finding deadlock is given in [40] using the theory of transitive matrix.

The above mentioned approaches can be applied for designing a deadlock-free policy, and the deadlock-handling problem is outside the scope of this paper, that mainly considers deadlock-free policy composition. Accordingly, we specify a deadlock free local policy as follows: let us suppose that a user requests a resource r_1 for executing an operation t_1. Before releasing r_1, the same user will request another resource r_2 for executing another operations t_2 (Fig. 9(a)). In this case, for the Petri net specification, both resources will be occupied by the user at the very beginning time of requesting r_1 (Fig. 9(b)). For instance, in the above example of policy composition, for the request of "printing and copying", in order to avoid system deadlock, transition $t_{1i}, i = 1, 2$ requests both printers and xeroxing machines even if printing and copying are not executed at the same time. We call such a way of requesting resources "complete occupying".

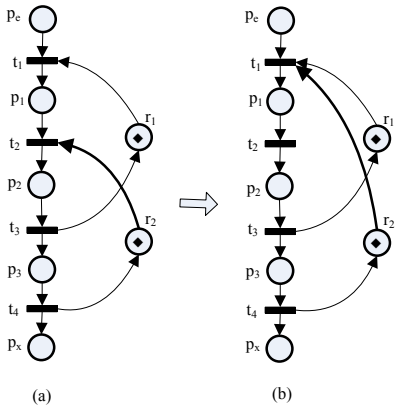

Fig. 9. Modification of sharing resources

With "complete occupying" in each local policy, each resource will be requested orderly without circular waiting and the composed system is deadlock free. However, the time duration of handling a request may be delayed. For instance, in the above printer accessing policy, let us assume that user A from domain D_1 requests printing and copying and user B from domain D_2 requests copying. Let us assume that there is only one printer and one xeroxing machine and in the EPNP specification, the time for printing is 10, for copying is 5, that is, $\tau(t_{21}) = \tau(t_{51}) = \tau(t_{52}) = 10, \tau(t_{31}) = \tau(t_{71}) = \tau(t_{72}) = 5$, and for firing other transitions is 0. Based on the global policy, the minimum time for handling these two requests is 15, that is, while user A is printing, user B can copy simultaneously. However, in the "complete occupying" case, since both the printer and the xeroxing machine are occupied by user A, user B has to wait until user A releases the xeroxing machine. In this case, the total time for handling the requests is 20.

Although "complete occupying" may delay the decision for a request, it prevents a policy from the difficult deadlock-handling problems. Searching for an optimal scheduling algorithm for fastly handling a request is out of scope of this paper.

Coming back to policy composition, we get the following conclusions.

Proposition 9: Let us consider a global policy $B = (P \cup R, T, F, W, M_0, C, \tau, p_e, p_x)$ composed from two local policies $B_i = (P_i \cup R, T_i, F_i, W_i, M_{i0}, C_i, \tau_i, p_{ie}, p_{ix})$ $(i = 1, 2)$ by applying place fusion with "complete occupying" resources. Then B is complete (resp., terminating, consistent, confluent) provided both B_1 and B_2 are complete (resp., terminating, consistent,

confluent).

Proof: Since the resources are shared by "complete occupying", B_1 and B_2 are executed independently. Each firing sequence of B is a union of sequences of B_1 and B_2. Correspondingly, each reachable marking in B is a union of reachable markings of B_1 and B_2. The proof about preserving policy properties is similar to Proposition 7. □

5.3 Policy Composition via Operation Synchronization

Let us now consider another kind of composition through an operation synchronization, first presented through an example.

Example 2 (writing accessing policy (WAP)): Let us consider two local policies, each one designed for writing some local documents. Each local policy is as follows: a specific user requests writing a document. Once the document is available, it can be written, then, after writing, the document is returned to its place. In the context of cooperation, due to security reasons, the global policy requires that some special documents in a set D (such as contracts) cannot be signed unless two specific users from different domains sign them together.

In this example, the global policy should be composed from the two local policies by applying transition fusion. It is formally defined as follows and, for understanding the definition, the reader can refer to Fig. 7 where the common resources set place R has to be replaced by the common transition set S.

Definition 18 (transition fusion): For two processes $B_i = (P_i, T_i \cup S, F_i, W_i, M_{i0}, C_i, \tau_i, p_{ie}, p_{ix})$ $(i = 1, 2)$, their composition by operation synchronization (applying transition fusion for S), denoted $B_1[S]B_2$, is defined as the process $B = (P, T \cup S, F, W, M_0, C, \tau, p_e, p_x)$, where $P = P_1 \cup P_2 \cup \{p_{1e}, p_{2e}, p_{1x}, p_{2x}\}$, p_e and p_x are newly added interface places; $T = T_1 \cup T_2 \cup S \cup \{t_e, t_x\}$; $F = F_1 \cup F_2 \cup \{(p_e, t_e), (t_e, p_{1e}), (t_e, p_{2e}), (p_{1x}, t_x), (p_{2x}, t_x), (t_x, p_x)\}$; $W = W_1 \cup W_2 \cup \{W(p_e, t_e), W(t_e, p_{1e}), W(t_e, p_{2e})\} \cup \{W(p_{1x}, t_x), W(p_{2x}, t_x), W(t_x, p_x)\}$; where $W(t_x, p_x)$ is a 2-dimension vector $(W(p_{1x}, t_x), W(p_{2x}, t_x))$ $M_0 = M_{10} \cup M_{20}$; $C = C_1 \cup C_2$; $\tau(t) = \begin{cases} \tau_i(t), t \in T_i; \\ max\{\tau_i(t)\}, t \in S. \end{cases}$

To illustrate the definition of transition fusion, let us consider the above Example 2. Fig. 10 gives the specification of the two local policies, where transition t_{1i} requests a document for processing; if some normal documents are requested, t_{2i} is fired for processing documents, if some special documents that belong to D are requested, t_2 is fired; after processing document, t_{3i} is fired for returning the documents $(i = 1, 2)$. After policy composition, the global policy requires that the normal documents are processed by local users, while for processing the special documents in D, two specific users from different domains should be both on the scene (on-line or off-line) and execute the processing operation together. Hence transitions t_2 in both local policies should be fused into a single transition t_2 in the global policy.

For policy composition via transition fusion, we have the following results.

Proposition 10: Let us consider a global policy $B = (P, T \cup S, F, W, M_0, C, \tau, p_e, p_x)$ composed from two local policies

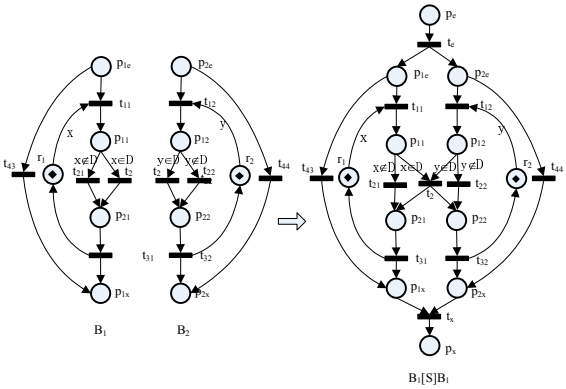

Fig. 10. Policy composition via transition fusion

$B_i = (P_i, T_i \cup S, F_i, W_i, M_{i0}, C_i, \tau_i, p_{ie}, p_{ix})$ $(i = 1, 2)$ by applying transition fusion. Then B is complete (resp., terminating, consistent, confluent) provided both B_1 and B_2 are complete (resp., terminating, consistent, confluent).

Proof: By transition fusion, the control flow in each local policy remains unchanged except that the total time duration of executing the policy is delayed. Each firing sequence of B is a union of sequences of B_1 and B_2. The proof about preserving policy properties is similar to Proposition 7. □

5.4 Policy Refinement

For security requirement, sometimes an encapsulated policy should be added to an existing policy. The former is named a sub-policy, whereas the latter is named a super-policy. This section discusses how to compose a sub-policy and a super-policy by applying place refinement and transition refinement based on EPNP specification.

The formal definition of composition via place refinement is as follows.

Definition 19: (Fig. 11) For two processes $B_i = (P_i, T_i, F_i, W_i, M_{i0}, C_i, \tau_i, p_{ie}, p_{ix})$ $(i = 1, 2)$, their composition by applying place refinement for place $p_r \in P_1$, denoted $B_1[p_r \mapsto B_2]$, is defined as the process $B = (P, T, F, W, M_0, C, \tau, p_e, p_x)$, where $P = P_1 \cup P_2 \cup \{p_{2e}, p_{2x}\} - \{p_r\}$, $p_e = p_{1e}$ and $p_x = p_{1x}$; $T = T_1 \cup T_2$; $F = F_1 \cup F_2 \cup \{({}^\bullet p_r, p_{2e}), (p_{2x}, p_r^\bullet)\} - \{({}^\bullet p_r, p_r), (p_r, p_r^\bullet)\}$; $W = W_1 \cup W_2 \cup W({}^\bullet p_r, p_{2e}) \cup W(p_{2x}, p_r^\bullet) - W({}^\bullet p_r, p_r) - W(p_r, p_r^\bullet)$; $M_0 = (M_{10} - \{M_{10}(p_r)\}) \cup M_{20} \cup M_0(p_{2e})$, where $M_0(p_{2e}) = M_{10}(p_r)$; $C = C_1 \cup C_2$; $\tau = \tau_1 \cup \tau_2$.

Based on the above definition, a sub-policy B_2 is inserted into a super-policy B_1 by refining place p_r with B_2. Those input (resp., output) transitions of p_r in B_1 become the input (resp., output) transitions of place p_{2e} (resp., p_{2x}) in B. The tokens of p_r in B_1 move to place p_{2e} in B. Other parts do not change (Fig. 11).

In order to illustrate the application of place refinement for policy composition, let us consider the following example.

Example 3 (document accessing policy): Let us assume that there are two types of documents distinguished as local document (L) and global document (D). The policy about accessing a document is as follows: for requesting a local

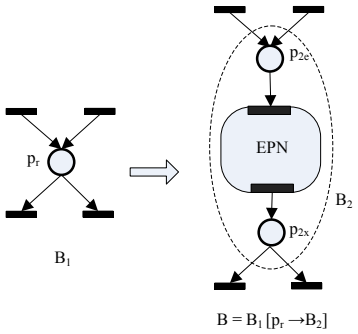

Fig. 11. Policy composition via place refinement

document, once the document is available, the request is permitted. While for requesting a global document, for security reason, it should be decided by applying Chinese Wall policy (CWP) whose detailed specification is introduced in Section 7.1). Hence, as a sub-policy, the Chinese Wall policy will be composed with the super-policy by applying the place refinement operator, This example will be furthe detailled in Section 7.

Let us now consider property preservation results for place refinement.

Proposition 11: Let a global policy $B = (P, T, F, W, M_0, C, \tau, p_e, p_x)$ be composed from two local policies $B_i = (P_i, T_i, F_i, W_i, M_{i0}, C_i, \tau_i, p_{ie}, p_{ix})$ $(i = 1, 2)$, by refining place $p_r \in P_1$ into B_2.

1) If B_1 is complete, B_2 is complete and consistent, then B is complete;
2) If both B_1 and B_2 terminate, so does B;
3) If both B_1 and B_2 are consistent, so is B;
4) If B_1 is confluent and B_2 terminates properly, then B is confluent.

Proof:

1) If B_2 is complete and consistent, B_2 will output an identical token each time it is referred. Consequently, the firing sequence in B_1 part will remain unchange and hence B is complete if B_1 is complete.
2) If B_2 terminates, B_2 may or may not output a token. Correspondingly, in B, the transitions in $p_r^\bullet$ may or may not be fired. As a result, each reachable state of B is a union of the states in B_1 and B_2. Obviously, B will terminate if B_1 can terminate.
3) If B_2 is consistent, B_2 either always outputs an identical token, or never outputs a token. In the first case, the firing sequences in B_1 part remain unchanged and hence B is consistent. If B_2 never outputs a token, then the transitions in $p_{2x}^\bullet$ are never fired in B. As a result, some firing sequences in B_1 part will never appear in B. But those firing sequences that do not include any transition of $p_{2x}^\bullet$ still exist in B. If B_1 is consistent, the control flow in B_1 part will not be changed and hence B is still consistent.
4) If B_2 terminates properly, B_2 will output a single token and resume to its static state M_{20} once it is referred. Each reachable marking of B is a union of the markings of B_1 and B_2. For any two reachable markings $M_i = P_{im} + Q_{im}$ in B, where P_{im}, Q_{im} are markings in B_1 and B_2 respectively and $i = 1, 2$, since B_1 is confluent and B_2 terminates properly, there exist $M_1' \in R(B_1, P_{1m}) \cap R(B_1, P_{2m})$ and $M_{20} \in R(B_2, Q_{1m}) \cap R(B_2, Q_{2m})$. Then $M = M_1' + M_{20} \in R(B, M_1) \cap R(B, M_2)$ and B is confluent. □

If an encapsulated policy is added by transition refinement, we will modify the super-policy by splitting the refined transition, and then place refinement operator can be applied for policy composition. The formal definition of transition splitting is defined as follows.

Definition 20: (Fig. 12) For a process $B = (P, T, F, W, M_0, C, \tau, p_e, p_x)$, splitting transition $t \in T$ results in the process $B' = (P', T', F', W', M_0', C', \tau', p_e', p_x')$, where $P' = P \cup \{p\}$, $T' = (T - \{t\}) \cup \{t_1, t_2\}$; $F' = (F - \{(^\bullet t, t), (t, t^\bullet)\}) \cup \{(^\bullet t, t_1), (t_1, p), (p, t_2), (t_2, t^\bullet)\}$; $W' = (W - \{W(^\bullet t, t), W(t, t^\bullet)\}) \cup \{W(^\bullet t, t_1), W(t_1, p), W(p, t_2), W(t_2, t^\bullet)\}$; $M_0' = M_0; C' = C, \tau' = \tau, p_e' = p_e, p_x' = p_x$.

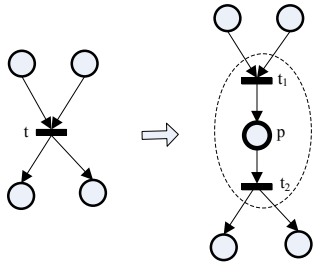

Fig. 12. Transition splitting

For transition refinement, it is easy to prove that the transition splitting operator preserves all the properties considered in this paper. Hence, we get the following results. The proof is the same as for Proposition 11.

Proposition 12: Let a global policy $B = (P, T, F, W, M_0, C, \tau, p_e, p_x)$ be composed from two local policies $B_i = (P_i, T_i, F_i, W_i, M_{i0}, C_i, \tau_i, p_{ie}, p_{ix})$ $(i = 1, 2)$, by applying transition refinement for transition $t \in T_1$.

1) If B_1 is complete, B_2 is complete and consistent, then B is complete;
2) If both B_1 and B_2 terminate, so does B;
3) If both B_1 and B_2 are consistent, so is B;
4) If both B_1 and B_2 are confluent, so is B.

Based on the composition operators defined in this section, a large security policy system can be specified step by step by composing its different modules. The previous propositions help verifying properties of a large system composed with these operators. However they have shown that the properties of confluence and consistency are not always preserved in such compositions. In the next section, we show how to restore in most cases these properties by a further composition with new EPNP closely related to XACML policy combiners.

6 EPNP-Based Specification of XACML Combiners

For a security system, in particular an access control system, the same resource may be requested by different policies and

their respective decisions may be different. This question has been largely addressed and a common technique is to use XACML policy combiners to solve the conflicts resulting from applying different policies for the same resources. There are four combiners described as follows:

- *Permit-overrides:* whenever one of the policies answers to a request with a "Permit" decision, the final authorization for the composed policy is "Permit". The policy will generate a "Deny" only in the case where at least one of the sub-policies returns "Deny", and all others return "NotApplicable" or "Indeterminate". When all sub-policy return "NotApplicable", the final output is "NotApplicable". The decision is "Indeterminate" if no sub-policy returns a decision, i.e. when unexpected errors occur in every evaluation attempt.
- *Deny-overrides:* the semantics are similar to *Permit-overrides*. The only difference is to exchange "Permit" and "Deny" in the above description.
- *First-applicable:* the final authorization coincides with the result of the first sub-policy which produces the decision "Permit" or "Deny"; if no sub-policy is applicable, then the decision is "NotApplicable"; if errors occur, then it is "Indeterminate".
- *Only-one-applicable:* the resulting decision will be "Permit" or "Deny" if the single policy that applies to the request generates one of these decisions. The result will be "NotApplicable" if all policies return such decision. The result is "Indeterminate" if more than one policy set returns a decision different from "NotApplicable".

In order to simplify the specification model, we assume in this paper that there are only two sub-policies and no error occurs in the combiners, so there are only three possible decisions, namely "Permit", "Deny", and "NotApplicable". Actually it would be not harder but only more technical to handle multiple sub-policies and an additional decision "Indeterminate".

Let us first consider the formal specification of Permit-overrides combiner based on extended Petri net processes. The Petri net structure given in Fig. 13 is defined as $POC = (p_{3e}, p_{3x}, T_c, F_c, W_c, M_c, C_c)$. Places p_{3e} and p_{3x} have three types of tokens colored with *p*, *d*, *n* respectively, representing the three different decisions. Weights are assigned to each arc. For example, in Fig. 13, the weight of the arc (p_{3e}, t_{pp}) is $(2, 0, 0)$: this means that firing transition t_{pp} requires at least two tokens colored with *p*, i.e., both sub-policies return the decision "Permit". After firing transition t_{pp}, the output is $(1, 0, 0)$, meaning that place p_{3x} gets a token colored with *p*, i.e., the final decision is "Permit".

For the Deny-overrides combiner (DOC), the specification is similar to the above defined Permit-overrides combiner, just exchanging "Permit" and "Deny", *p* and *d*.

The First-applicable combiner $FAC = (\{p_{4e}, p_{4x}\}, T_c, F_c, W_c, M_c, C_c)$ is defined in Fig. 14, where the entry place p_{4e} represents all possible decisions taken by the sub-policies, while the exit place p_{4x} represents the final decision of the composed policy. T_c is the transition set: for each transition, there is a firing condition assigned to it. For example, the weight $(1, 0, 1)$ means that the decisions of the

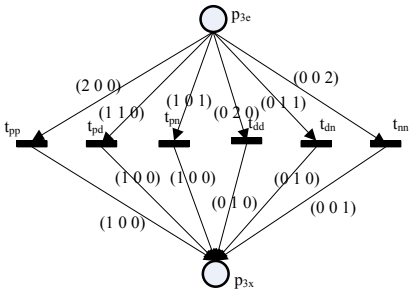

Fig. 13. Permit-overrides combiner POC.

sub-policy are "Permit" and "NotApplicable" respectively. The condition $\sigma_p \leq \sigma_n$ means that the firing sequence of σ_p is shorter than the sequence σ_n, so the first decision is "Permit". Note that σ_p and σ_n are firing sequences of the two sub-policies. Hence the final decision is made based on not only the sub-policies' decisions but also their firing sequences.

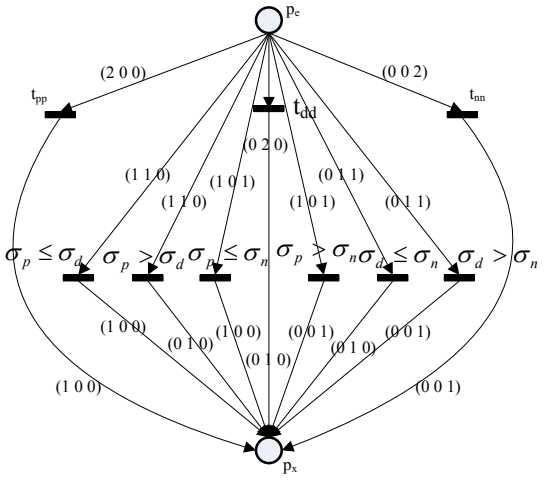

Fig. 14. First-applicable combiner FAC.

For the only-one-applicable combiner, the extended Petri net module $OAC = (\{p_{5e}, p_{5x}\}, T_c, F_c, W_c, M_c, C_c)$ is shown in Fig. 15. It contains only two transitions: if there exists only one possible decision of "Permit" $(1, 0, 0)$ or "Deny" $(0, 1, 0)$ or there are two decisions of "NotApplicable" $(0, 0, 2)$, the transition t_d can be fired and the place p_{5x} gets a same colored token. Otherwise transition t_n is fired and p_{5x} outputs a decision "NotApplicable".

Proposition 13: POC, DOC, FAC, OAC are EPNP which are complete, strongly terminating, consistent and confluent.

Proof: For each of POC, DOC, FAC and OAC, there are only two reachable markings, one is the initial marking and the other is the exit marking. All the firing sequences contains only one transition in T_c. It is obvious that they are complete, strongly terminating, consistent and confluent. □

As a consequence of Propositions 5 and 13, if *B* is complete, strongly terminating, consistent and confluent, and if *COM* is

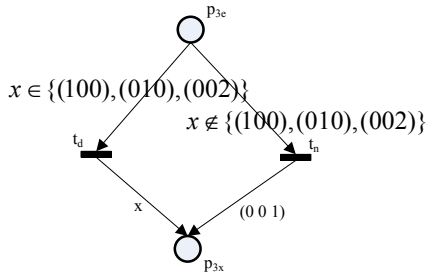

Fig. 15. Only-one-applicable combiner OAC.

one of POC, DOC, FAC, OAC, $B >> COM$ will enjoy the same properties.

Moreover, when B is itself built from other modules using composition operators $[], |||, [>$, but is not confluent or not consistent, one of the previous combiner can be used to restore these properties.

Proposition 14: Suppose B is composed from B_1 and B_2 by applying one of Choice, Interleave and Disable operators and COM is any one of the combiners POC, DOC, OAC and FAC.

1) $B >> COM$ is complete, strongly terminating if B_1 and B_2 enjoy these two properties.
2) Let $B = B_1[]B_2$. $B >> COM$ is consistent if B_1 and B_2 always output the same tokens; $B >> COM$ is confluent if both B_1 and B_2 terminate properly.
3) Let $B = B_1|||B_2$ and $COM \in \{POC, DOC, OAC\}$. Then $B >> COM$ is consistent and confluent if B_1 and B_2 enjoy these properties.
   If $B = B_1|||B_2$ and $COM = FAC$, then $B >> COM$ is confluent. $B >> COM$ is consistent provided that for all reachable markings M_1, M_2 that satisfy $M_{10}[B_1, \sigma_1\rangle M_1$, $M_{20}[B_2, \sigma_2\rangle M_2$, $M_1(p_{1x}) > 0$, $M_2(p_{2x}) > 0$, we always have either $\sigma_1 \leq \sigma_2$ or $\sigma_2 \leq \sigma_1$.
4) Let $B = B_1[> B_2$, then $B >> COM$ is consistent provided that 1) COM = FAC; 2) B_1 is consistent and for all reachable markings M_1, M_2 that satisfy $M_{10}[B_1, \sigma_1\rangle M_1$, $M_{20}[B_2, \sigma_2\rangle M_2$, $M_1(p_{1x}) > 0$, $M_2(p_{2x}) > 0$, we always have either $\sigma_1 \leq \sigma_2$ or $\sigma_2 \leq \sigma_1$.
   $B >> COM$ is confluent if B_1 and B_2 are confluent and satisfy ${}^\bullet(P_d^\bullet) = \{p_d\}$, $P_d^\bullet = {}^\bullet p_{2x}$, and $W(P_d, {}^\bullet p_{2x}) = W(P_d, T_d), W({}^\bullet p_{2x}, p_{2x}) = W(T_d, p_{2x})$.

Proof:

1) By Propositions 6, 7 and 8, B preserves these two properties. By Propositions 5 and 13, $B >> COM$ also preserves these two properties.
2) If $B = B_1[]B_2$, when B_1 and B_2 output the same tokens, then the input of COM is the same and hence the final decision is consistent. If both B_1 and B_2 terminate properly, for any two reachable markings, they will reach the exit marking of $B >> COM$ and hence $B >> COM$ is confluent.
3) If $B = B_1|||B_2$, since both B_1 and B_2 are consistent, their output is always the same and hence the input for the entry place of COM is always the same, and COM is consistent by Proposition 13. By Propositions 5 and 7, $B >> COM$ is confluent.
4) If $B = B_1[> B_2$ and the condition is satisfied, then the final decision is the same as that of B_1 and hence consistent. Based on Propositions 5, 8 and 13, B is confluent and so is $B >> COM$.

□

7 A Complex Policy Design — case study

Let us consider a situation where a user requests access to documents belonging to different competitive companies. Such access is granted or denied on the basis on a Chinese Wall Policy [41]; if the user has a reading access to a document, he may print it and/or copy it. If he has a writing access, he may modify it for instance by signing the document.

The information flow is as follows: a user requests "reading" or "writing" a document according to the Chinese Wall Policy (CWP); in the first case, once the access decision for reading is obtained, the user can continue processing the document by applying the "printer accessing policy" (PAP); in the second case, once the access decision for writing is obtained, the user can process the document by applying the "writing access policy" (WAP).

For cooperation, the two domains are combined. Their printers and xeroxing machines are shared by users from both domains. Generally, their local documents can be handled based on their local security policies. But let us assume that some special documents, in a set D, can be processed only if specific users from both domains handle them together.

Based on our approach, a policy design has the following three steps: specification of primitive modules; composition of sub-policies; and verification of policy correctness. In the following subsections, we introduce the specification and verification of both local and global policies.

7.1 Specification of primitive modules

The principles of sub-policies PAP and WAP have been described in Section 5.2, and Section 5.3 respectively. The Petri net based specifications of these policy modules are shown in Figure 8 and Figure 10 respectively. We concentrate in the following on the description of CWP.

Chinese wall policy is about preventing the conflict of interest between clients. Figure 16 is an example of context where application of the Chinese Wall policy [41] is relevant. The objects of the database contain the information related to companies; a company dataset (CD) contains objects related to a single company; a conflict of interest (COI) class contains the datasets of companies in competition. For example, the bank COI class contains three competitive companies (i.e., three CDs). The read and write policies in Chinese wall policy are defined as follows [41]:

Read policy: a subject $s \in S$ can read an object $o \in O$ provided that, either there is an object $o' \in O$ such that s has accessed o' and $CD(o') = CD(o)$, or for all objects $o', o' \in PR(s) \Rightarrow COI(o') \neq COI(o)$, where $PR(s)$ is the set of object s has accessed previously.

In other words, a subject s is permitted to read an object o provided that, either s reads the objects all in the same CD, or reads the objects in different COIs. In the same COI, the subject cannot read objects in different CDs.

Write policy: a subject $s \in S$ may write to an object $o \in O$ provided that s is permitted to read o, and for all the objects o', s can read $o' \Rightarrow CD(o') = CD(o)$.

In other words, a subject s is permitted to write an object o only when s can read o and other objects accessible by s are in the same CD with o.

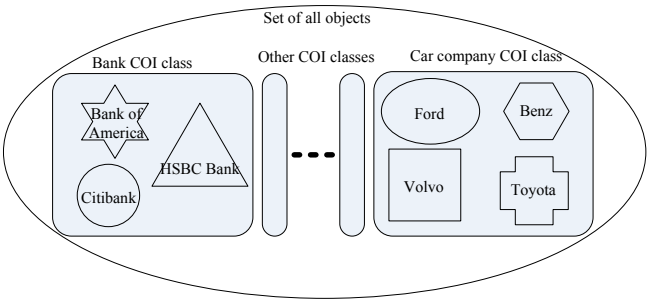

Fig. 16. An example for Chinese wall policy.

The Bank COI class consists of three CDs, i.e., Bank of America, HSBC Bank and Citibank (Figure 16). The data in the same CD will be represented with the same color, and the three CDs are specified with three different colors a, b, and c, respectively (Figure 17). Once there is a request, e.g., request for an object in Bank of America, then a token a is put into the entry place and the EPNP model is initially marked. Since requesting two different CD is not permitted, the entry place is initially marked with only one token. For simplicity, we denote the marking and weight in the EPNP with colors instead of vectors. For example, in Figure 17, we use a to replace vector $(a, 0, 0)$ and use x, y and z to denote any one of the colors.

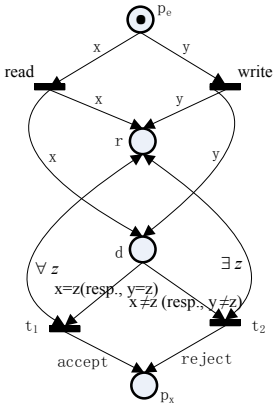

Fig. 17. The specification of one COI class of the Chinese wall policy.

When the subject requests to read objects in some CD, the transition *read* is considered. Place r is a record place: once the subject had a request, the corresponding data will have a record in this place. Since firing transition t_1 and t_2 will not consume the tokens in place r, it is possible to specified record in this way. Suppose a subject had previously requested some objects. Based on the Chinese Wall policy, if the new request belongs to the same CD as before, the new request is granted. In the EPNP specification, if all data recorded in place r belong to the same color, i.e., $\forall z, x = z$, transition t_1 is firable and the output is "Accept"; otherwise, $\exists z, x \neq z$, then transition t_2 is firable and the output is "Reject".

When the subject requests to write objects in some CD, the transition *write* is considered and the request data is recorded in place r. Based on the Chinese Wall policy, once the new writing-requested object is in a different CD from previous requests, the request will be rejected. In the EPNP specification, if $\exists z \neq y$, transition t_2 is firable and the output is "Reject"; otherwise, $\forall z, y = z$, transition t_1 is firable and the output is "Accept".

7.2 Composition of sub-policies

For handling a local document (including the common documents D), both local policy and global policy contains some sub-policies, that are Chinese Wall policy (CWP), printer accessing policy (PAP) and writing accessing policy (WAP). Based on the information flow, the global EPNP specification is shown in Figure 18. The sub-policies are combined by applying Enable and Choice operators based on the global policy requirement.

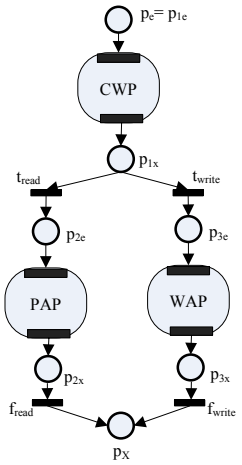

Fig. 18. Abstracted specification of the policy (for both local and global use).

7.3 Local policy specification and verification

The detailed EPNP specification of the local policy is shown in Figure 19. Let us explain it.

For a local request, the EPNP-based specification of CWP for one COI is presented in Section 7.1.

In case of a reading request, transition t_{read} is fired and, provided the decision of CWP was "(read, accept)", the user can request copying the document (firing transition t_{71}), or printing the document (firing transition t_{51}), or printing and copying (firing transition t_{11}). Once there exists an available

xeroxing machine (i.e. when r_2 is marked), the user can copy. After copying, transition t_{81} is fired and the xeroxing machine is released; once there exists an available printer ((i.e. when r_1 is marked), the user can print. After printing, transition t_{61} is fired and the printer is released; once there exist available printers and xeroxing machines, the user can print and copy; then transitions t_{31} and t_{41} are fired and both resources are released. If the decision of CWP was "(read, reject)", transition t_{91} is fired. The final transition f_{read} just outputs "OK" , meaning that the request has been handled.

In case of a writing request, transition t_{write} is fired and, provided the decision of CWP was "(write, accept)", the user can write the document. When the document is available (i.e. when place r_3 is marked), t_{13} is firable and the user can process writing the document (t_{23} corresponds to the operation of writing); after writing, the document is released and updated (t_{33} is fired). If the decision of CWP was "(write, reject)", transition t_{93} is fired. The final transition f_{write} just outputs "OK" , meaning that the request has been handled.

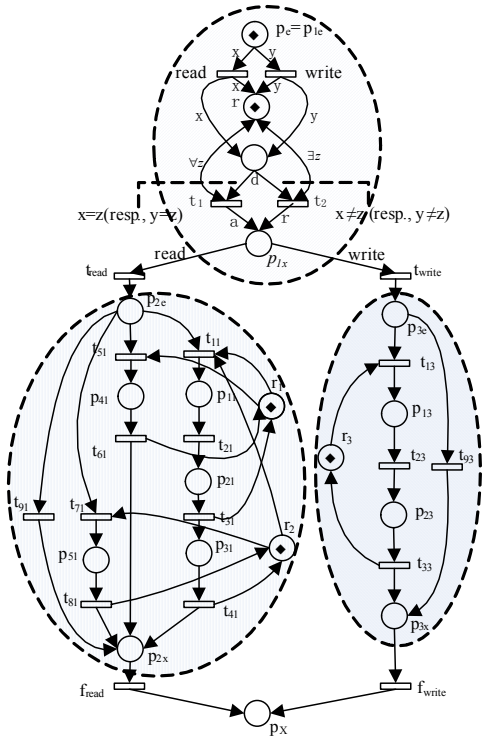

Fig. 19. EPNP specification of the local policy.

The local policy (Figure 19) consists of three sub-policies, namely CWP, PAP, WAP which are combined by applying Enable and Choice operators. Based on Propositions 5 and 6, the abstract EPNP model shown in Figure 19 is correct, i.e., complete, terminating, consistent and confluent. It is easy to verify that all three sub-policies are correct. At the same time, both PAP and WAP terminate properly. Although CWP does not terminate properly (place *r* is unbounded), the control flow is similar to that of a properly terminating process. Hence, after adding the three sub-policies to the abstract EPNP by applying place refinement, the resulting local policy (Figure 19) is correct (based on Proposition 11).

7.4 Global policy specification and verification

The global policy specification is given in Figure 20). Let us explain how it is built, in comparison with the local policy specification.

For CWP, in an interoperation domain, the policy has to deal with multiple COIs. Hence, it differs from the local policy by changing the conditions of firing transitions t_1 and t_2. Based on Chinese Wall policy, for a new request "reading $x \in (CD \in COI_i)$", there are two cases for accepting the request: the first is when all its recorded data z belong to the same CD as x, that is both x and z have the same color ($x = z$); another case is when x belongs to a COI diffrent from its records, i.e., $x \in COI_i, z \in COI_j$. Otherwise, i.e., $\exists z \neq x, x, z \in COI_i$, the request is rejected. As for a request "writing y", only when all the records and x belong to the same CD, i.e., $\forall z, y = z$, the request is accepted. Otherwise, i.e., $\exists z \neq y$, the request is rejected.

For PAP, the difference is sharing resources r_1 and r_2, which can be specified by place fusion with complete occupying resources.

For WAP, if the document does not belong to *D*, transition t_{23} (resp., t_{24}) can be fired just as processing local documents. If the document belongs to *D*, then the document should be processed by two users from different domains. Hence, the fused transition t_2 is fired and the document is processed by two users together at the same time. Here transition t_2 represents the document processing, which is assigned with time constraints. Transition t_{43} (resp., t_{44}) is used for specifying bypass: when a local document is processed, the cooperating domain can be bypassed directly by firing transition t_{43} or t_{44}.

The global policy (Figure 20) consists of five sub-policies, namely one CWP with multiple COI records, two PAPs, and two WAPs. The two PAPs are combined by applying place fusion with complete occupying resources, and the two WAPs are combined by applying transition fusion. Then, based on the super-policy, i.e., the global EPNP (Figure 18), the three (combined) sub-policies are added by applying place refinement operator. It is easy to verify that each primitive policy is correct. By Proposition 9, Proposition 10, the two combined sub-policies PAP and WAP are both correct. At the same time, both PAP and WAP terminate properly. Although CWP does not terminate properly (place *r* is unbounded), the control flow is similar to that of a properly terminating process. Hence, after refining the global EPNP by applying place refinement, the resulted global policy (Figure 20) is correct, i.e., complete, terminating, consistent and confluent (based on Proposition 11). Note that, in this example, the sub-policies belong to different types. For instance, CWP will output a decision, while PAP and WAP will not. PAP may result in deadlock if resource competition exists, while WAP never be in such a situation. Theoretically, once the user's request of "reading" or "writing" is permitted, the user succeeds in copying, printing or writing for the document. However, in order to avoid deadlock, it is possible for a user to wait printers and copy machines for a very long time if many new requests keep on coming and occupying the resources.

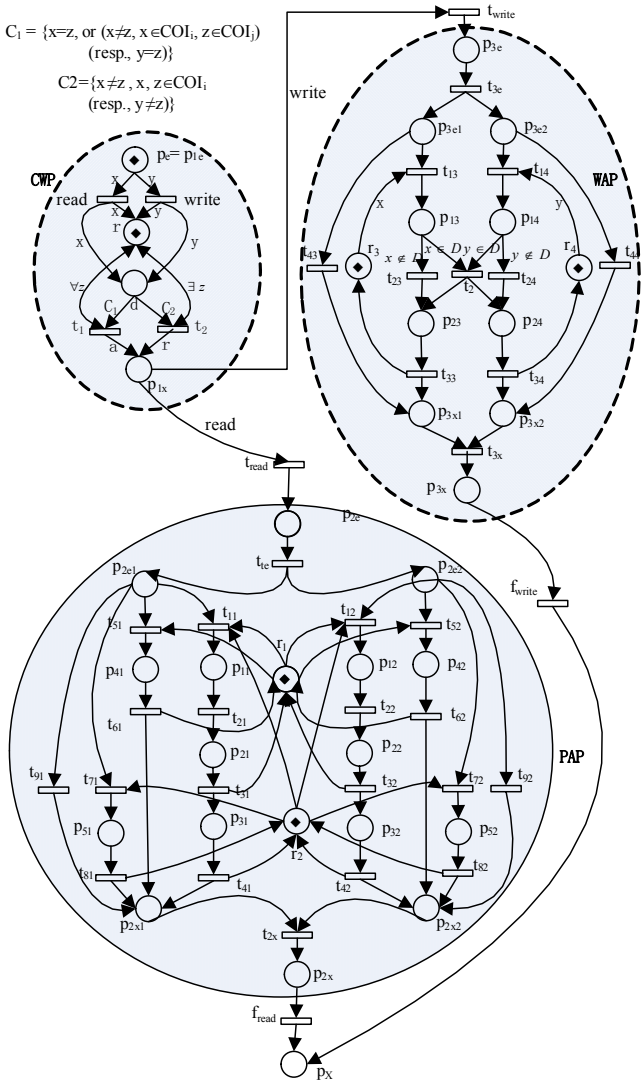

Fig. 20. EPNP specification of the global policy.

8 Conclusive remarks

In an extended Petri net, time is assigned to transitions for specifying the duration of executing operations, colors are assigned to tokens and weights for distinguishing different data, resources and the preconditions of executing operations. Based on the newly defined concept of extended Petri net process, this paper specifies security policies in a modular way. Policy composition operators are specified and property preserving results are stated for verification. This technology is suitable for general security policy design, especially for large and complex security systems. In a real-life software system, the system requirements may be changeable, system action is dynamic and the security policy is complex. Hence, much more policy composition operators should be available for the specification, and much more policy properties should be defined and verified for satisfying the policy requirements. In future work, the focus is on looking for new modeling techniques, composition operators and formal definition and verification of policy properties.

Because of the complexity and large-scale of real life systems, the policy properties may not always be preserved properly when applying PPPA. For example, consistency and confluence properties cannot be preserved even by adding a XACML combiner for the Disable operator. In order to design a safe policy, we may try to design a new combiner for composition in order to restore the policy properties for the resultant policy. The combiner may not be based only on the decisions of sub-policies, but possibly also on the activity of the sub-policies. Designing new combiners for restoring the unpreserved properties is another interesting future research topic.

References

[1] Jajodia, S., Samarati, P., Sapino, M.L., Subrahmanian, V.S.: Flexible support for multiple access control policies. ACM Trans. Database Syst. **26**(2) (2001) 214–260

[2] Kalam, A., Baida, R., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miege, A., Saurel, C., Trouessin, G.: Organization based access control. Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on (2003) 120–131

[3] Moses, T.: Extensible access control markup language (XACML) version 2.0. Technical report, OASIS (February 2005)

[4] Huang, H.J.: Enhancing the Property-Preserving Petri Net Process Algebra for Component-based System Design (with Application to Designing Multi-agent Systems and Manufacturing Systems). PhD thesis, Department of Computer Science, City University of Hong Kong (2004)

[5] Mak, W.: Verifying Property Preservation for Component-based Software Systems (A Petri-net Based Methodology). PhD thesis, Department of Computer Science, City University of Hong Kong (2001)

[6] Dougherty, D.J., Kirchner, C., Kirchner, H., Santana de Oliveira, A.: Modular access control via strategic rewriting. In: Proceedings of 12th European Symposium On Research In Computer Security (ESORICS'07), Dresden (Sep 2007)

[7] Bonatti, P.A., di Vimercati, S.D.C., Samarati, P.: An algebra for composing access control policies. ACM Trans. Inf. Syst. Secur. **5**(1) (2002) 1–35

[8] Wijesekera, D., Jajodia, S.: A propositional policy algebra for access control. ACM Trans. Inf. Syst. Secur. **6**(2) (2003) 286–325

[9] Bauer, L., Ligatti, J., Walker, D.: Composing security policies with polymer. In: PLDI. (2005) 305–314

[10] Lee, A.J., Boyer, J.P., Olson, L., Gunter, C.A.: Defeasible security policy composition for web services. In Winslett, M., Gordon, A.D., Sands, D., eds.: FMSE, ACM (2006) 45–54

[11] Bruns, G., Dantas, D.S., Huth, M.: A simple and expressive semantic framework for policy composition in access control. In: FMSE '07: Proceedings of the 2007 ACM workshop on Formal methods in security engineering, New York, NY, USA, ACM (2007) 12–21

[12] Santana de Oliveira, A.: Rewriting-based access control policies. Electronic Notes in Theoretical Computer Science **171**(4) (2007) 59–72

[13] Kirchner, C., Kirchner, F., Kirchner, H.: Strategic computations and deductions. Studies in Logic and the Foundations of Mathematics.Festchrift in honor of Peter Andrews (2008)

[14] Santana de Oliveira, A.: Réécriture et modularité pour les politiques de sécurité. PhD thesis, UHP Nancy 1 (2008)

[15] Shafiq, B., Masood, A., Joshi, J., Ghafoor, A.: A role-based access control policy verification framework for real-time systems. In: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. (2005)

[16] Mortensen, K.H.: Automatic code generation method based on coloured petri net models applied on an access control system. Lecture Notes in Computer Science **1825** (2000) 367–386

[17] Knorr, K.: Dynamic access control through petri net workflows. In: Proceedings of 16th Annual Conference on Computer Security Applications. (2000) 159–167

[18] Zhang, Z.L., Hong, F., Liao, J.G.: Modeling chinese wall policy using colored petri nets. In: Proceedings of the 6th IEEE International Conference on Computer and Information Technology. (2006)

[19] Zhang, Z.L., Hong, F., Xiao, H.: Verification of strict integrity policy via petri nets. In: Proceedings of the International Conference on Systems and Networks Communication. (2006)

[20] Juszczyszyn, K.: Verifying enterprise's mandatory access control policies with coloured petri nets. In: Proceedings of the 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. (2003)

[21] Deng, Y., Wang, J.C., Tsai, J., Beznosov, K.: An approach for modeling and analysis of security system architectures. IEEE Transactions on Knowledge and Data Engineering **15**(5) (2003) 1099–1119

[22] Murata, T.: Petri nets: Properties, analysis, and applications. Proceedings of IEEE **77**(4) (1985) 541–580

[23] Tschantz, M.C., Krishnamurthi, S.: Towards reasonability properties for access-control policy languages. In Ferraiolo, D.F., Ray, I., eds.: SACMAT, ACM (2006) 160–169

[24] Barker, S., Fernández, M.: Term rewriting for access control. In: DBSec. (2006) 179–193

[25] Tiplea, F., Jucan, T., Masalagiu, C.: Term rewriting systems and petri nets. Analele Stiintifice ale Universitatii Al. I. Cuza **34**(4) (1988) 305–317

[26] Verma, R., Rusinowitch, M., Lugiez, D.: Algorithms and reductions for rewriting problems. Fundamental Informatics **46**(3) (2001) 257–276

[27] Leahu, I., Tiplea, F.: The confluence property for petri nets and its applications. In: Proceedings of the 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. (2006)

[28] Melinte, R., Oanea, O., Olga, I., Tiplea, F.: The home marking problem and some related concepts. Acta Cybernetica **15**(3) (2002)

[29] Xing, K.L., Jin, X.J., Feng, Y.: Deadlock avoidance petri net controller for manufacturing systems with multiple resource service. Proc. IEEE Conference on Robotics and Automation (2005) 4757–4761

[30] Lee, J.K.: Scheduling analysis with resources share using the transitive matrix based on p-invariant. Proc. 41st SICE Annual Conference **2** (2002) 1359–1364

[31] Li, Z., Zhou, M.: Elementary siphons of petri nets and their application to deadlock prevention in flexible manufacturing systems. IEEE Transactions on System, Man, and Cybernetics **34** (2004) 38–51

[32] Hu, H., Li, Z., Wang, A.: On the optimal set of elementary siphons in petri nets for deadlock control in fms. Proc. 2006 IEEE International Conference on Networking, Sensing and Control (2006) 244–247

[33] Li, Z., Wei, N.: Deadlock control of flexible manufacturing systems via invariant-controlled elementary siphons of petri nets. The International Journal of Advanced Manufacturing Technology **33** (2007) 24–35

[34] Roszkowska, E.: Supervisory control for deadlock avoidance in compound processes. IEEE Transactions on System, Man, and Cybernetics **34** (2004) 52–64

[35] Xu, G., Wu, Z.M.: A kind of deadlock-free scheduling method based on petri net. Proc. IEEE HASR'02 (2002) 195–200

[36] Xu, G., Wu, Z.M.: Deadlock-free scheduling method using petri net model analysis and ga search. Proc. 2002 International Conference on Control Application **2** (2002) 1153–1158

[37] Liu, J., Itoh, Y., Miyazawa, I., Sekiguchi, T.: A research on petri net properties using transitive matrix. Proc. IEEE International Conference on System, Man, and Cybernetics (1999) 888–893

[38] Song, Y.J., LEE, J.K.: Deadlock analysis of petri nets using the transitive matrix. Proc. 41st SICE Annual Conference (2002)

[39] Kim, S., Lee, S., Lee, J.: Deadlock analysis of petri nets based on the resource share places relationship. IMACS multiconference on CESA (2006) 59–64

[40] Lee, J.: Deadlock find algorithm using the transitive matrix. Proc. CIE'04 (2004)

[41] Zhang, Z., Hong, F., Liao, J.: Modeling chinese wall policy using colored petri nets. Proceedings of the 6th IEEE International Conference on Computer and Information Technology (2006)